MULTI-THREADED FRAME SAFE SYNCHRONIZATION
OF A SIMULATION

Technical Field

The present invention relates to simulations and, in particular, to a

5   simulation in which the various aspects of the simulation are executed

asynchronously but communicate with each other in a safely synchronized

manner.

Background

Virtual prototyping solutions can greatly accelerate product

10   development. Obviously, the more accurately a solution simulates "reality",

the more confidence one can have in that solution. On the other hand,

though, the most useful simulation is typically the one that not only

simulates reality faithfully, but also takes the fewest resources to do so.

Whereas reality is continuous in nature, computer simulations are

15   forced to approximate the continuity by acting upon and examining and/or

evaluating the simulation model only at discrete points in time. The greater

the frequency at which the simulation model is examined, the more

accurate is the simulation, since continuity is more closely approached.

Thus, conventionally, simulations are designed such that the evaluation rate

20   is high enough to sufficiently approximate reality (i.e., with acceptable error

and without noticeable jumps in display), but not so fast that the

computer's resources are completely exhausted.

The evaluation rate required depends largely upon the rate at which

the simulated events would occur in reality. For example, if modeling the

25   fast-moving parts within an engine, the evaluation rate should be relatively

high. On the other hand, modeling the slow buildup of heat within the

engine could be carried out at a much slower rate. Furthermore, where visual interaction is important, evaluation for display should be at a relatively high rate if the sensation of interactivity is to be provided to the user.

5   Since different aspects of a simulation occur with varying evaluation rate requirements, it would be preferable if each aspect executed at a rate most suitable to it. Unfortunately, conventional simulations force the entire simulation to execute at a common frequency. The common frequency is generally faster than required for some aspects of the simulation and

10   perhaps slower than ideally required for other aspects of the simulation. For example, if the calculations of engine heat buildup could only be performed at once or twice a second due to the complexity of the calculations, the calculation of engine part positions could only be performed one or twice a second as well, even though it would be a much

15   simpler calculation. Furthermore, visualization of movement around the engine would be seriously hampered, with position updates at only the once or twice per second rate.

     A typical solution to the evaluation rate problem discussed above is to perform the simulation in batch, allowing the user to inspect the results

20   in detail once the simulation is completed. A disadvantage of this approach, however, is that the simulation cannot be interactive.

Summary

     In accordance with the invention, a computer program is provided to

25   execute a simulation. The computer program includes a plurality of service program means, where the services are configured to collectively determine simulated attributes of objects of an environment under simulated

operation. Each service is associated with at least a subset of object
attributes in an object database. Furthermore, each service is executing at
a rate independent of the other services, and at least some of the services
access and operate upon object attributes, from the object database, with
5   which the service is associated.

A write queue program is associated with each service, to queue
write requests from the service to write determined simulated attributes to
the object database. Nodes coordinate execution of the queued requests to
cause the determined simulated attributes to be written to the object
10   database in a manner such that each service has a coherent view of all the
object attributes.

Brief Description of the Figures

Fig. 1 broadly illustrates a simulation environment 100 in accordance
15   with an embodiment of the invention.

Fig. 2 illustrates processing within a service of the Fig. 1
environment 100.

Fig. 3 illustrates how a node change request is handled by a node.

Fig. 4 illustrates how a node processes a request from a service to
20   ensure the service is pointing to the most recently determined value for an
object (or portion of an object).

Fig. 5 is a time line that broadly illustrates how services interact.

Fig. 6 is a time line that illustrates the interaction between services
in greater detail.

Detailed Descripti n

   A simulation environment 100 is now described, first broadly with reference to Fig. 1. Block 102 is an object database. The "attributes" of "objects" in the database 102 represent the "state" of the environment

5 being simulated. Block 104 illustrates the execution of "services" (in this case, services A though C) that constitute the active (computational) part of the simulation.

   Block 106 illustrates a "data probe context". Input and output data probes are computational objects (implemented as services in the service

10 execution block 104) that provide the simulation user an interface to attributes of objects in the object context 102. Input data probes inject values from some source (e.g., a file or a connection to another application) into attributes of an object during a simulation. Conversely, output data probes examine the values of attributes during a simulation and provide

15 those values to some destination such as a file or a connection to another application. Data probes are described later in greater detail.

   Finally, block 108 illustrates a "behavior context". Behavior is the programming employed by a service (in the service execution block 104) that defines how a service is to affect the modeled environment during a

20 simulation. One type of behavior is "logic", which is an event-based state machine description of behavior. Another type of behavior is "mechanism", which is a physics, kinematics-based description of behavior.

   Now, the operation of the services execution block 104 is described in greater detail. As alluded to above, services are objects that perform

25 calculations or other functionality repeatedly to accomplish the simulation. Typically, multiple services exist simultaneously and the services each perform their functionality at a particular temporal frequency. Theoretically, the number of services executing at one time is unlimited,

although as a practical matter the number of services may be limited by the operating system or by computational performance limitations.

In accordance with an embodiment of the invention, the multiple services operate asynchronously, but data communication among the

5   multiple asynchronously operating services is safely synchronized. This is accomplished even while accommodating the various asynchronous "frame rates" of the services.

To support the multiple "views" of object data required by various services. All objects to which access is required by a service are derived

10   from a "node" base class. The node interacts with services to provide frame safe access to its data members. Each node may maintain multiple "node images" for an object (or for a portion of an object). Each service potentially sees different sets of changes at various times, although all services are ultimately working on the same values. In one embodiment, to

15   minimize the impact on memory, the data duplication occurs at a relatively fine granularity, within objects referred to as "nodes". Nodes are typically relatively small units (portions of object), and data copies are maintained only for those nodes requiring them. Thus, memory duplication is minimized.

20   In one embodiment, a node does not actually contain attribute data itself. Rather, a node serves as a wrapper to one or more node images that contain the attribute data. While a node image actually contains data, a node includes "frame-safe" methodology to provide the services access the data in the node images. That is, in essence, a node hides the "multi-

25   service issues" from the services. A service that needs to access the value of an object attribute requests the value from or writes the value to, a node corresponding to that object (or to the portion of the object that includes the attribute).

The service passes an ID of the service as an argument to the node. The node uses the ID internally to manage multiple node images it may contain. The node determines which node image is accurate for the requesting service, and creates and deletes node images as needed to

5   maintained the required "clean" copies.

Before discussing in greater detail how a service accesses node data, though, it is useful to briefly discuss the "phases" of processing that a service executes. Fig. 2 illustrates the processing phases of a typical service. First, the service performs its computations in a computational

10   phase 202. During this computational phase 202, the service queues up (but does not yet process) "change requests". A "change request" is a request to a node to change an attribute of an object represented by the node. Also, during the computational phase 202, the service requests reading of attribute values required for its processing. The requests to read

15   attribute values are not queued but, rather, are processed as part of the computational phase 202.

Next, the service enters a change request processing phase 204 in which it actually processes (or causes to be processed) the change requests queued up during the computational phase 202. In one embodiment, the

20   change request processing by the service consists simply of looping through the queued change requests and passing them to the appropriate nodes for execution. Alternately, the service could notify the nodes (or some other program) that change requests are in its queue for processing.

Finally, if the time period for this frame of the service has not

25   expired by the end of the change request processing phase 204, the service enters an idle phase 206 during which it does nothing substantive until it is time to enter the computational phase 202 again.

The processing of change requests is now described in greater detail. Fig. 3 illustrates the processing performed by a node in response to a change request passed from a service. After starting (302), at step 304 it is determined if the node image corresponding to the service is the "latest

5   image" or if another service is referencing this node. What is meant by "latest image" is that this node image represents the latest updated state of the object represented by the node. (The process by which a node image becomes the "latest image" is described later.)

If either the node image is the latest image or a service other than

10   the requesting service is referencing the node, then at step 306 a new node image is created. Also, the node records that the latest updated state of the object represented by the node (i.e., "latest image") is this new node image. The old node image is deleted.

Then (or if the determination at step 304 is "NO"), at step 308 other

15   services for which the node is maintaining node images are notified that the "latest image" has changed. At step 310, the requested write operation is made to the node image for the service (whether the "old" node image corresponding to the service or the newly created node image corresponding to the service).

20   The notification (step 308), and how the notified services react to a notification, is now described. Referring back to Fig. 2, a notification is processed by a service as a "null" change request during the change request processing step 204. In particular, in response to such a notification, the notified service creates a change request for the node for

25   which has been changed by the notifying service. The change request is "null", though, because it does not include data to be written to the object represented by the node. Rather, the change request is a request to make

the node image associated with the notified service corresponds to the "latest image" if required.

Fig. 4 illustrates how a node processes a "null" change request. First, the processing starts at step 402. At step 404, a determination is made as to whether the node image presently associated with the calling service (i.e., the notified service) is the "latest" node image. If not, then there is no further processing of the "null" change request. If the node image presently associated with the calling service is not the "latest" node image, then the node associates the calling service with the "latest" node and then deletes the "old" node image for the calling service.

Having described the operation of a service in isolation and also how a node reacts to change requests by a service, reference is now made to Figs. 5 and 6 to describe how services executing in accordance with the previous discussion coexist in apparent synchronicity. Fig. 5 is a timeline that shows how each service executes in its own "frame time". In Fig. 5, the actual time a particular service spends in a frame differs from the time spent in a frame by another service, although this isn't shown in Fig. 5. That is, the services are not synchronized as would appear from Fig. 5 on its face. In fact, though, as is shown later with reference to Fig. 6, it appears to each service that it is in fact operating synchronously with the other services.

Turning now to Fig. 5, and looking at Frame N first, each service reads object attributes from an object "frame context" 502 that appears to that service to be synchronous with the frame execution of the service. (Remembering the previous discussion, this is accomplished in one embodiment by the service generating a read request to a node object, and the node object providing the requested attribute data from a "latest" node image associated with the node.) During its computational phase (202,

-8-

also shown in Fig. 2), the service queues "change requests" (indicated in Fig. 5 as WriteOp's) to write object attribute data back to the object context 502. After the computational phase, each service processes its queued change requests during a protected critical section by asking the

5　appropriate nodes to perform the changes indicated by the change requests (as shown in Fig. 3). In one embodiment, during the "critical section" for processing a change to a node, the node may not otherwise be accessed, either for read access or for write access. Fig. 5 further illustrates similar processing occurring in Frame N+1.

10　　　　Now, Fig. 6 is a timeline in "real time" that illustrates how the node processing provides the appearance of synchronization to the services. Taking service A first, it can be seen (as denoted by arrow 602a) that Service A is provided a node attribute (in shorthand, "reads" the attribute, even though such "reading" is via the node object) for processing. At some

15　point later (denoted by arrow 604a), Service A queues a write operation of an object attribute. Yet later (denoted by arrow 606a), during a protected "critical" section of processing, Service A causes (requests) the appropriate node to write the object attribute to an image (the current – or "latest" – image). This is similarly occurring for Service B and Service C, shown in

20　Fig. 6 using the same reference numerals as for Service A but with subscripts B and C, respectively.

　　　　That is, Service B reads object attributes (602b); computes on the object attributes and queues write operations of object attributes (604b); and causes the object attributes of the queued write operations to be

25　associated with the current image for the node (606b). A similar processing occurs for Service C (602c, 604c and 606c).

　　　　It should be noted that the invention is to be defined by the claims, and not by the embodiment(s) just described.